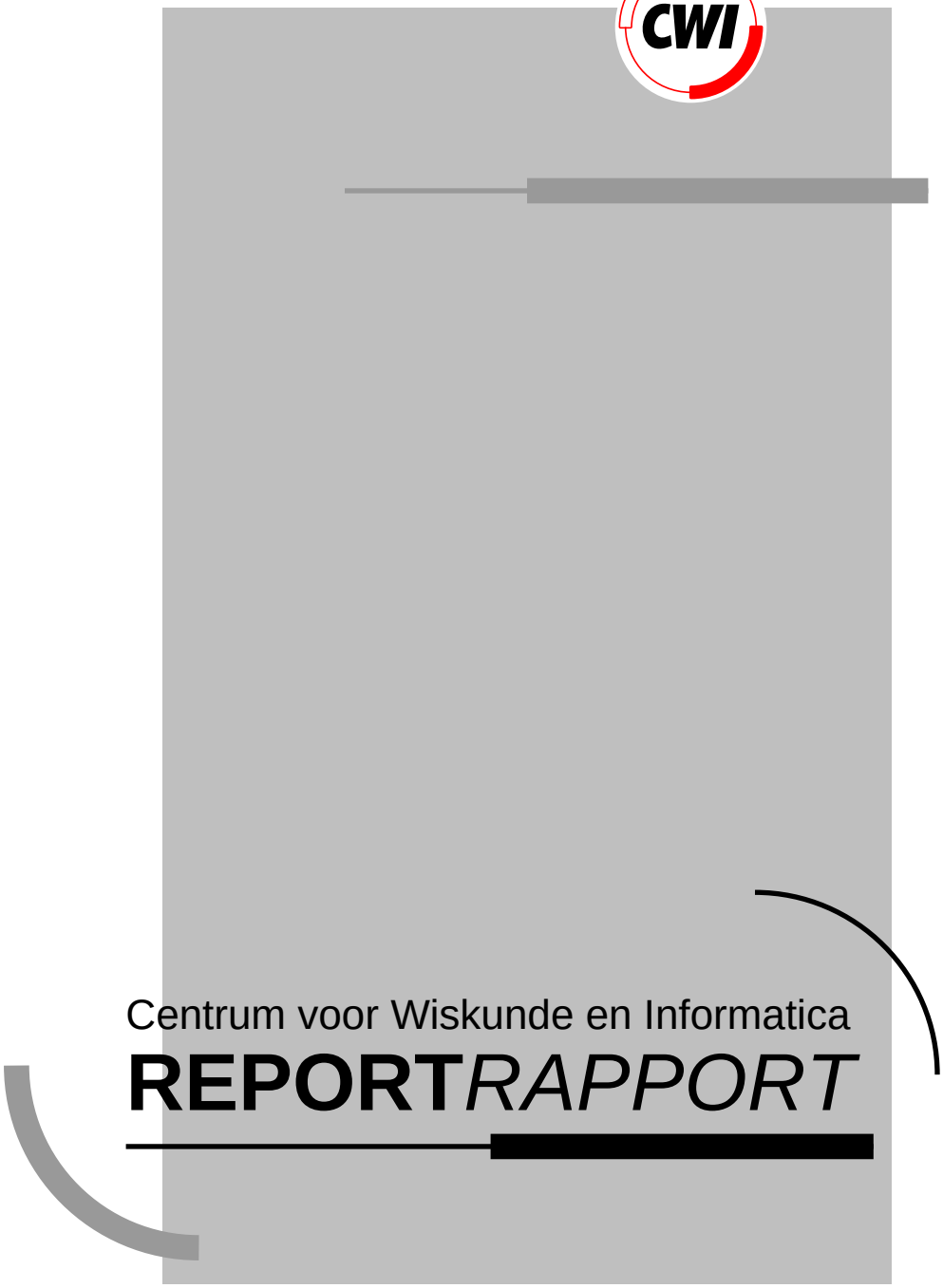

Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Supervisory control for nondeterministic systems

A. Overkamp

Department of Operations Reasearch, Statistics, and System Theory

BS-R9411 1994

Supervisory Control for Nondeterministic Systems

Ard Overkamp

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

In this paper we present the first results of our attempt to set up a supervisory theory for nondeterministic discrete event systems. The supervisory control problem for nondeterministic systems is specified and motivated. A necessary and sufficient condition for the existence of a supervisor is derived. Also an algorithm is presented that will automatically generate a supervisor. The results are extended to deal with uncontrollable events.

AMS Subject Classification (1991): 68Q75, 93B25

Keywords & Phrases: Nondeterministic Discrete Event Systems, Failure Semantics, Supervisory Control.

1. Introduction

Up till now discrete event systems have been mostly modelled as deterministic systems. The current state and the next event uniquely determine the next state. Deterministic systems are effectively described by the language they can generate. Conditions under which a supervisor exists and the behavior of the controlled system are also stated in a language context. But in case of partial observation (and partial specification) systems are more appropriately modelled as nondeterministic systems.

It is not sufficient to describe nondeterministic systems only by the language they generate. The blocking properties of the systems are also important [5]. Consider the following example. Given two systems, A and B (Fig. 1 a, b). A is nondeterministic and B is its deterministic equivalent. The languages that the systems can generate are the same, but when connected to system C, (Fig. 1 c) $A||C$ can deadlock but $B||C$ can not (Fig. 1 d, e).

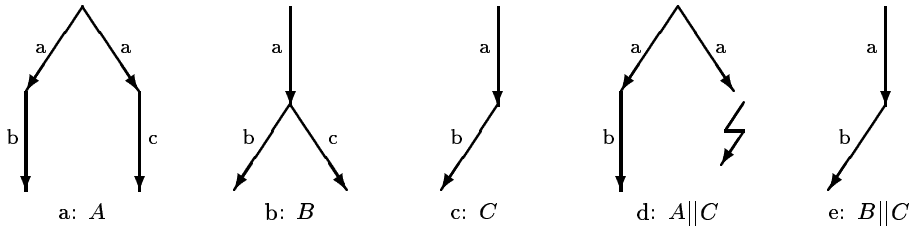

Figure 1: Illustration of the difference between a nondeterministic and a deterministic system.

The problem that we are trying to solve in this paper is the following. Given a nondeterministic system and a nondeterministic specification, find a supervisor such that the controlled system behaves according to the specification. Here 'behaves like' means that the controlled system may only do what the specification allows, and that it may only block an event if the specification can also block that event. A new aspect in this setup is that the specification may also be nondeterministic. This gives the possibility to specify that some nondeterminism is still allowed in the controlled system.

In Section 3 an alternative for language semantics will be introduced: failure semantics [1, 4]. It is a commonly used semantics in computer science. We will use it to state necessary and sufficient conditions for the existence of a supervisor such that the controlled system behaves according to the

specification. An algorithm to automatically generate a supervisor will be presented. In Section 5 these results will be extended to deal with uncontrollable events.

2. Notation and Definitions

Definition 2.1 (Finite State Machine) A (nondeterministic) finite state machine (FSM) is a four tuple $(Q, \Sigma, \delta, \mathcal{Q}_0)$ where,
Q is the state space,
Σ is the set of event labels,
$\delta : Q \times \Sigma \to 2^Q$ is the transition function, and
$\mathcal{Q}_0 \subseteq Q$ is the nonempty set of initial states.

The domain of the transition function can be extended in a natural way to strings and sets of states. Let $A = (Q_\mathrm{a}, \Sigma, \delta_\mathrm{a}, A_0)$ be a FSM, $\mathcal{A} \subseteq Q_\mathrm{a}$, $q_\mathrm{a} \in Q_\mathrm{a}$, $\sigma \in \Sigma$, and $s \in \Sigma^*$. We define

- $\delta_\mathrm{a}(q_\mathrm{a}, \varepsilon) = q_\mathrm{a}$,
- $\delta_\mathrm{a}(q_\mathrm{a}, s\sigma) = \delta_\mathrm{a}(\delta_\mathrm{a}(q_\mathrm{a}, s), \sigma)$,
- $\delta_\mathrm{a}(\mathcal{A}, s) = \bigcup_{q \in \mathcal{A}} \delta_\mathrm{a}(q, s)$.

For notational convenience we will use the state machine itself as an argument instead of the initial states. $\delta(A, s) = \delta_\mathrm{a}(A_0, s)$.

For a deterministic finite state machine (DFSM) $\delta(q, s)$ is always a singleton set. In this case we will write $q' = \delta(q, s)$ instead of $\{q'\} = \delta(q, s)$. Also, we will simply write the initial state instead of a singleton set containing the initial state.

A useful function is the out-function, which describes the outgoing arcs of a state. $\mathrm{out}(q) = \{\sigma \in \Sigma |\, \delta(q, \sigma) \neq \emptyset\}$.

The set of traces, or language, generated by a FSM is defined as $L(A) = \{s \in \Sigma^* |\, \delta(A, s) \neq \emptyset\}$. A language is prefix closed if $sv \in L \Rightarrow s \in L$. All languages generated by a FSM are prefix closed.

Control will not be enforced by a control map as in the Ramadge-Wonham approach, but by synchronization on common events. The controlled system (i.e. the synchronous composition of the plant and the supervisor) can only execute those events that both the supervisor and the plant can execute.

Definition 2.2 (Synchronous Composition) Let $A = (Q_\mathrm{a}, \Sigma, \delta_\mathrm{a}, A_0)$ and $B = (Q_\mathrm{b}, \Sigma, \delta_\mathrm{b}, B_0)$. The synchronous composition of A and B is $A||B = (Q_\mathrm{a} \times Q_\mathrm{b}, \Sigma, \delta_\mathrm{ab}, A_0 \times B_0)$ where $\delta_\mathrm{ab}((q_\mathrm{a}, q_\mathrm{b}), \sigma) = \delta_\mathrm{a}(q_\mathrm{a}, \sigma) \times \delta_\mathrm{b}(q_\mathrm{b}, \sigma)$. Note that for $\mathcal{Q}_\mathrm{a} \subseteq Q_\mathrm{a}, \mathcal{Q}_\mathrm{b} \subseteq Q_\mathrm{b} : \emptyset \times \mathcal{Q}_\mathrm{b} = \mathcal{Q}_\mathrm{a} \times \emptyset = \emptyset$.

The next lemma is a direct result from the definition.

Lemma 2.3 *For FSMs A, B, and $s \in \Sigma^*$;*

1. $(q_\mathrm{a}, q_\mathrm{b}) \in \delta(A||B, s) \;\Leftrightarrow\; q_\mathrm{a} \in \delta(A, s) \wedge q_\mathrm{b} \in \delta(B, s)$.
2. *For* $(q_\mathrm{a}, q_\mathrm{b}) \in \delta(A||B, s)$ $\mathrm{out}((q_\mathrm{a}, q_\mathrm{b})) = \mathrm{out}(q_\mathrm{a}) \cap \mathrm{out}(q_\mathrm{b})$.
3. $L(A||B) = L(A) \cap L(B)$.

3. Failure Semantics

In the introduction we already argued that the language alone does not provide enough information for nondeterministic systems. The blocking properties of the systems are also important. In computer science failure semantics is introduced to deal with such problems. In failure semantics a system is not only described by the traces it can generate but also by the set of events that it can block after a trace (fail to execute). We will not elaborate on it here. The reader is referred to [1, 4] for more information. We will only define what we need and show that it fits our interests.

First, let us define what we mean by blocking. In the specification it may be stated that after certain traces the system is allowed to block, but after other traces it may not. So we have to define nonblockingness with respect to an already executed trace.

DEFINITION 3.1 (BLOCKING) Let A be a FSM and $s \in L(A)$; A is nonblocking after s if $\forall q_a \in \delta(A,s)$ $\text{out}(q_a) \neq \emptyset$. A FSM A is nonblocking if it is nonblocking after all $s \in L(A)$.

As stated in the introduction, the controlled system may only do what the specification allows and it may only block an event if the specification can also block that event. Because the specification is nondeterministic it is possible that sometimes an event is blocked in one branch of a nondeterministic choice and allowed in another branch. So, the controlled system has the choice to either block or execute that event. Because of this freedom the system does not have to be equivalent to the specification. It only has to reduce (implement) it [2, 3].

DEFINITION 3.2 (REDUCTION) Let A, B be FSMs; A reduces B ($A \sqsubseteq_{\approx} B$) if
i) $L(A) \subseteq L(B)$, and
ii) $\forall s \in L(A)$, $\forall q_a \in \delta(A,s)$, $\exists q_b \in \delta(B,s)$ s.t. $\text{out}(q_b) \subseteq \text{out}(q_a)$.

Here, point *i* states that system A may only do what system B allows, and point *ii* states that A may only block what B can also block.

The reduction relation guarantees that the controlled system does not block in an environment if the specification does not block in the same environment. This is formally stated in Theorem 3.3.

THEOREM 3.3 *Let A, B be FSMs such that $L(A) \subseteq L(B)$. For all FSM C and $s \in \Sigma^*$, $B||C$ nonblocking after s implies $A||C$ nonblocking after s, if and only if $A \sqsubseteq_{\approx} B$.*

Proof (if part) if $A \sqsubseteq_{\approx} B$ then $\forall q_a \in \delta(A,s)$ $\exists q_b \in \delta(B,s)$ s.t. $\text{out}(q_b) \subseteq \text{out}(q_a)$. If $B||C$ is nonblocking after s then, by Lemma 2.3, $\forall q_c \in \delta(C,s)$ $\text{out}(q_b) \cap \text{out}(q_c) \neq \emptyset$. Then also $A||C$ is nonblocking after s because $\text{out}(q_a) \cap \text{out}(q_c) \supseteq \text{out}(q_b) \cap \text{out}(q_c) \neq \emptyset$.

(only if part). We will prove that if A does not reduce B then there exists a FSM C and a string s such that $B||C$ is nonblocking after s, but $A||C$ can block after s. If not $A \sqsubseteq_{\approx} B$ then $\exists s \in L(A)$, $\exists q_a \in \delta(A,s)$ s.t. $\forall q_b \in \delta(B,s)$ $\text{out}(q_b) \not\subseteq \text{out}(q_a)$. Let C be a DFSM such that $q_c = \delta(C,s)$ with $\text{out}(q_c) = \Sigma - out(q_a)$. Then $\forall q_b \in \delta(B,s)$ $\text{out}(q_b) \cap \text{out}(q_c) = \text{out}(q_b) \cap (\Sigma - \text{out}(q_a)) = \text{out}(q_b) - \text{out}(q_a) \neq \emptyset$. But $\text{out}(q_a) \cap \text{out}(q_c) = \text{out}(q_a) \cap (\Sigma - \text{out}(q_a)) = \emptyset$. □

4. CONTROLLER SYNTHESIS

In this section we will first state under what condition there exists a supervisor such that the controlled system reduces the specification. We will call this condition reducibility. Then, an algorithm will be presented that generates such a supervisor. In the following G will denote an uncontrolled system and E a specification.

DEFINITION 4.1 (REDUCIBILITY) Let G, E be FSMs. A language K is reducible (w.r.t. G, E) if

$$\forall s \in K, \forall q_g \in \delta(G,s), \exists q_e \in \delta(E,s) \text{ s.t. } \text{out}(q_e) \subseteq \text{out}(q_g) \cap \{\sigma \in \Sigma | s\sigma \in K\}.$$

LEMMA 4.2 *Let G, E be FSMs. Let S be a DFSM such that $L(S)$ is reducible. Then $G||S \sqsubseteq_{\approx} E$.*

Proof (Point i of the definition of reduction) If $s \in L(S) \cap L(G)$ then $\delta(G,s)$ is not empty. So, by the definition of reducibility, there exists a $q_e \in \delta(E,s)$. Hence $s \in L(E)$, and $L(G||S) \subseteq L(E)$.

(Point ii of the definition of reduction). For $s \in L(G||S)$ let $(q_g, q_s) \in \delta(G||S, s)$, then, by Lemma 2.3, $q_g \in \delta(G,s)$ and $q_s = \delta(S,s)$. Because S is deterministic $\{\sigma \in \Sigma | s\sigma \in L(S)\} = \text{out}(q_s)$. Then, by reducibility of $L(S)$, we know $\exists q_e \in \delta(E,s)$ such that $\text{out}(q_e) \subseteq \text{out}(q_g) \cap \text{out}(q_s)$. Hence also, by Lemma 2.3, $\text{out}(q_e) \subseteq \text{out}((q_g, q_s))$. □

THEOREM 4.3 *Let G, E be FSMs. There exists a supervisor S such that $G||S \sqsubseteq_{\approx} E$ if and only if there exists a nonempty, prefix closed, and reducible language K.*

Proof (if part) Let S_{K} be a deterministic state machine generating K. The proof follows directly from Lemma 4.2.

(only if part). Let $K = L(S||G)$. Then K is nonempty and prefix closed. We will prove that K is reducible. $\forall s \in K,\ s \in L(S||G)$, so $\delta(S||G, s) \neq \emptyset$. Then, by lemma 2.3, $\delta(G, s) \neq \emptyset$ and $\delta(S, s) \neq \emptyset$. Thus $\forall s \in K,\ \forall q_{\mathrm{g}} \in \delta(G, s)\ \exists q_{\mathrm{s}}$ s.t. $(q_{\mathrm{g}}, q_{\mathrm{s}}) \in \delta(G||S, s)$. By the definition of reduction $\exists q_{\mathrm{e}} \in \delta(E, s)$ s.t. $\mathrm{out}(q_{\mathrm{e}}) \subseteq \mathrm{out}((q_{\mathrm{g}}, q_{\mathrm{s}}))$. By lemma 2.3 $\mathrm{out}((q_{\mathrm{g}}, q_{\mathrm{s}})) \subseteq \mathrm{out}(q_{\mathrm{g}})$. And by construction of K $\mathrm{out}((q_{\mathrm{g}}, q_{\mathrm{s}})) \subseteq \{\sigma \in \Sigma | s\sigma \in K\}$. So, $\mathrm{out}(q_{\mathrm{e}}) \subseteq \mathrm{out}(q_{\mathrm{g}}) \cap \{\sigma \in \Sigma | s\sigma \in K\}$. □

ALGORITHM 4.4 The following algorithm will construct a supervisor (if it exists) such that the controlled system reduces the specification.

1. Generate a deterministic state machine $R^0 = (Q^0_{\mathrm{r}}, \Sigma, \delta^0_{\mathrm{r}}, R^0_0)$, where

$$Q^0_{\mathrm{r}} = 2^{Q_{\mathrm{g}}} \times 2^{Q_{\mathrm{e}}},$$

$$R^0_0 = (G_0, E_0),$$

$$\delta^0_{\mathrm{r}}((\mathcal{G}, \mathcal{E}), \sigma) = \begin{cases} (\delta_{\mathrm{g}}(\mathcal{G}, \sigma), \delta_{\mathrm{e}}(\mathcal{E}, \sigma)), & \text{if } \delta_{\mathrm{g}}(\mathcal{G}, \sigma) \neq \emptyset \wedge \delta_{\mathrm{e}}(\mathcal{E}, \sigma) \neq \emptyset, \\ \text{empty}, & \text{otherwise}. \end{cases}$$

2. Construct R^{i+1} from R^i by removing all reachable states $q^i_{\mathrm{r}} = (\mathcal{G}, \mathcal{E})^i$ (including its in- and out-going transitions) from the state space of R^i that do *not* satisfy the following condition,

$$\forall q_{\mathrm{g}} \in \mathcal{G}\ \exists q_{\mathrm{e}} \in \mathcal{E} \text{ s.t. } \mathrm{out}(q_{\mathrm{e}}) \subseteq \mathrm{out}(q_{\mathrm{g}}) \cap \mathrm{out}(q^i_{\mathrm{r}}) \ .$$

3. Repeat the previous step until all reachable states satisfy the condition or until there are no more reachable states left. Let R be the last R^i.

THEOREM 4.5 *Let G and E be FSMs. Then Algorithm 4.4 produces a DFSM R in a finite number of steps. If the state space of R is nonempty then $G||R \sqsubseteq_{\sim} E$. If the initial state is removed from R then no supervisor exists such that the controlled system reduces the specification.*

Proof (The algorithm stops in a finite number of steps) If in one step of the algorithm no state is removed, then the algorithm stops because all states satisfy the condition. If in every step at least one state is removed from the state space then the algorithm will eventually halt because the state space is finite.

(The algorithm returns a correct solution). We will proof that when the algorithm finds a solution then $L(R)$ is reducible. And thus, by Lemma 4.2, $G||R \sqsubseteq_{\sim} E$. Let $s \in L(R)$ and $q_{\mathrm{r}} = (\mathcal{G}, \mathcal{E}) = \delta(R, s)$. Then $\mathcal{G} = \delta(G, s)$, $\mathcal{E} = \delta(E, s)$, and because R is deterministic $\mathrm{out}(q_{\mathrm{r}}) = \{\sigma \in \Sigma | s\sigma \in L(R)\}$. Then, by step 3 of the algorithm, $\forall q_{\mathrm{g}} \in \delta(G, s)\ \exists q_{\mathrm{e}} \in \delta(E, s)$ s.t. $\mathrm{out}(q_{\mathrm{e}}) \subseteq \mathrm{out}(q_{\mathrm{g}}) \cap \mathrm{out}(q_{\mathrm{r}}) = \mathrm{out}(q_{\mathrm{g}}) \cap \{\sigma \in \Sigma | s\sigma \in L(R)\}$.

(The algorithm finds a solution if one exists). Assume there exists a nonempty, prefix closed, and reducible language K. First, we will prove that $K' = K \cap L(G)$ is also nonempty, prefix closed and reducible. Then we will prove that $K' \subseteq L(R)$. Thus the algorithm will return a nonempty solution.

Because $\varepsilon \in K$ and $\varepsilon \in L(G)$, $K \cap L(G) \neq \emptyset$. Because K and $L(G)$ are prefix closed, $sv \in K \cap L(G) \Rightarrow s \in K \wedge s \in L(G) \Rightarrow s \in K \cap L(G)$. So K' is prefix closed. By reducibility of K, $\forall s \in K \cap L(G)$, $\forall q_{\mathrm{g}} \in \delta(G, s)$, $\exists q_{\mathrm{e}} \in \delta(E, s)$ s.t. $\mathrm{out}(q_{\mathrm{e}}) \subseteq \mathrm{out}(q_{\mathrm{g}}) \cap \{\sigma \in \Sigma | s\sigma \in K\} = \mathrm{out}(q_{\mathrm{g}}) \cap \{\sigma \in \Sigma | s\sigma \in K \cap L(G)\}$. So K' is also reducible.

Now, we will prove by induction on the number of steps of the algorithm that $K' \subseteq L(R^i)$ for all i. So, also $K' \subseteq L(R)$.

Initial step: $\forall s \in K', s \in L(G)$. Then, by reducibility, $\delta(E, s) \neq \emptyset$. So $s \in L(E)$, and $K' \subseteq L(G) \cap L(E) = L(R^0)$.

Inductive hypothesis: $K' \subseteq L(R^i)$.
Let $s \in K'$ and $q_{\mathrm{r}}^i = \delta(R^i, s)$. By the hypothesis, $s\sigma \in K' \Rightarrow s\sigma \in L(R^i) \Rightarrow \delta(R^i, s\sigma) = \delta_{R^i}(q_{\mathrm{r}}^i, \sigma) \neq \emptyset \Rightarrow \sigma \in \mathrm{out}(q_{\mathrm{r}}^i)$. So, $\{\sigma \in \Sigma | s\sigma \in K'\} \subseteq \mathrm{out}(q_{\mathrm{r}}^i)$. By reducibility, $\forall q_{\mathrm{g}} \in \mathcal{G}$, $\exists q_{\mathrm{e}} \in \mathcal{E}$ s.t. $\mathrm{out}(q_{\mathrm{e}}) \subseteq \mathrm{out}(q_{\mathrm{g}}) \cap \{\sigma \in \Sigma | s\sigma \in K'\} \subseteq \mathrm{out}(q_{\mathrm{g}}) \cap \mathrm{out}(q_{\mathrm{r}}^i)$. So, q_{r}^i will not be removed from R^i. Hence, $s \in L(R^{i+1})$ and $K' \subseteq L(R^{i+1})$. □

From the last part of the proof it can be deduced that the algorithm generates the least restrictive supervisor. (The supremal element with respect to language inclusion).

5. Uncontrollable Events

Sometimes a system can generate events that can not be blocked by a supervisor (e.g. machine breakdown). Ramadge and Wonham showed that in the presence of these uncontrollable events we need the condition of controllability to guarantee the existence of a supervisor. We will show that for nondeterministic systems the same condition is needed.

Recall from [6] the definition of controllability. Let G be a FSM, $\Sigma_{\mathrm{u}} \subseteq \Sigma$. A prefix closed language K is controllable (w.r.t G, Σ_{u}) if $K\Sigma_{\mathrm{u}} \cap L(G) \subseteq K$. Note that this is equivalent to: $\forall s \in K$, $\forall q_{\mathrm{g}} \in \delta(G, s)$ $\mathrm{out}(q_{\mathrm{g}}) \cap \Sigma_{\mathrm{u}} \subseteq \{\sigma \in \Sigma | s\sigma \in K\}$.

Ramadge and Wonham called a supervisor that always accepts an uncontrollable event complete. We have to adapt the definition of completeness to deal with nondeterministic systems and control by synchronization.

Definition 5.1 (Completeness) A supervisor S is complete (w.r.t a FSM G) if

$$\forall s \in L(S||G),\ \forall q_{\mathrm{s}} \in \delta(S, s),\ \forall q_{\mathrm{g}} \in \delta(G, s),\ \mathrm{out}(q_{\mathrm{g}}) \cap \Sigma_{\mathrm{u}} \subseteq \mathrm{out}(q_{\mathrm{s}}) \ .$$

Theorem 5.2 *Let G and E be FSMs. There exists a supervisor S such that $G||S \sqsubseteq_{\lesssim} E$, and S complete w.r.t G if and only if there exists a nonempty, prefix closed, reducible and controllable language K.*

Proof (if part) Let S_{K} be a DFSM generating K. Then, by lemma 4.2, $G||S_{\mathrm{K}} \sqsubseteq_{\lesssim} E$. $\forall s \in L(S_{\mathrm{K}}||G)$, let $q_{\mathrm{s}} = \delta(S_{\mathrm{K}}, s)$. Then, by controllability of $L(S_{\mathrm{K}})$, $\forall q_{\mathrm{g}} \in \delta(G, s)$ $\mathrm{out}(q_{\mathrm{g}}) \cap \Sigma_{\mathrm{u}} \subseteq \{\sigma \in \Sigma | s\sigma \in L(S_{\mathrm{K}})\} = \mathrm{out}(q_{\mathrm{s}})$. so S_{K} is complete.

(only if part). Take $K = L(S||G)$, then by the proof of Theorem 4.3 (only if part) K is reducible. $\forall s \in L(S||G), \forall q_{\mathrm{s}} \in \delta(S, s), \forall q_{\mathrm{g}} \in \delta(G, s)$ $\mathrm{out}(q_{\mathrm{g}}) \cap \Sigma_{\mathrm{u}} \subseteq \mathrm{out}(q_{\mathrm{s}}) \Rightarrow \mathrm{out}(q_{\mathrm{g}}) \cap \Sigma_{\mathrm{u}} \subseteq \mathrm{out}(q_{\mathrm{s}}) \cap \mathrm{out}(q_{\mathrm{g}}) = \mathrm{out}((q_{\mathrm{g}}, q_{\mathrm{s}})) \subseteq \{\sigma \in \Sigma | s\sigma \in L(S)\}$. So, $L(S||G)$ is controllable. □

Algorithm 5.3 This algorithm constructs a complete supervisor (if it exists) such that the controlled system reduces the specification. The algorithm is the same as Algorithm 4.4, except that the following is added to step 2.

2. ...
   Also, remove those states $q_{\mathrm{r}}^i = (\mathcal{G}, \mathcal{E})^i$ that do *not* satisfy the following condition,

$$\forall q_{\mathrm{g}} \in \mathcal{G}\ \ \mathrm{out}(q_{\mathrm{g}}) \cap \Sigma_{\mathrm{u}} \subseteq \mathrm{out}(q_{\mathrm{r}}^i) \ .$$

Theorem 5.4 *Let G and E be FSMs. Then Algorithm 5.3 produces a DFSM R in a finite number of steps. If the state space of R is nonempty then $G||R \sqsubseteq_{\lesssim} E$ and R is complete. If the initial state is removed from R then no complete supervisor exists such that the controlled system reduces the specification.*

Proof The proof goes along the same lines as the proof of Theorem 4.5. The following has to be added to the different steps of the proof.

(The algorithm returns a correct solution). $\forall s \in L(R||G)$, let $q_r = (\mathcal{G}, \mathcal{E}) = \delta(R, s)$. Then, by step 3 of the algorithm, $\forall q_g \in \mathcal{G} = \delta(G, s)$, $\text{out}(q_g) \cap \Sigma_u \subseteq \text{out}(q_r)$. Hence R is complete.

(The algorithm finds a solution if one exists). After the sentence that starts with 'By reducibility': Also, by controllability, $\forall s \in K', \forall q_g \in \mathcal{G}$, $\text{out}(q_g) \cap \Sigma_u \subseteq \{\sigma \in \Sigma | s\sigma \in K'\} \subseteq \text{out}(q_r^i)$. So, q_r^i will not be removed from R^i ... □

6. Conclusions

In case of systems that are partially observed or partially specified one has to realize that the behaviour of a system depends on the nondeterministic properties of that system. This paper has been an attempt to set up a supervisory theory for nondeterministic systems. A condition (reducibility) is found for the existence of a supervisor such that the controlled system behaves like the specification. An algorithm is described which synthesizes a deterministic least restrictive supervisor. These results are extended to deal with uncontrollable events.

What remains to be done is to analyze the consequences of these results for systems with partial specification and partial observation.

References

1. J.C.M Baeten and W.P. Weijland. *Process algebra*. Cambridge University Press, Cambridge, 1990.
2. E. Brinksma, G. Scollo and C. Steenbergen. LOTOS specifications, their implementations, and their tests. *Proc. of IFIP Workshop 'Protocol Specification, Testing and Verification VI'*, pages 349–360, 1987.
3. S.D. Brooks, C.A.R. Hoare, and A.W. Roscoe. A theory for communicating sequential processes. *Journal of the ACM*, 31:560–599, 1984.
4. R.J. van Gladbeek. The linear time - branching time spectrum. *Proc. CONCUR '90*, Amsterdam, Lecture Notes in Computer Science 458, pages 278–297, 1990.
5. M. Heymann. Concurrency and discrete event control. *IEEE Control Systems Magazine*, 10:103–112, 1990.
6. P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77:81–98, 1989.